Improving Scalability and Maintenance of Software for High-Performance Scientific Computing by Combining MDE and Frameworks

Marc Palyart[1,2], David Lugato[1], Ileana Ober[2], and Jean-Michel Bruel[2]

[1] CEA / CESTA
33114 Le Barp - France
{marc.palyart,david.lugato}@cea.fr
[2] IRIT – Université de Toulouse
118, route de Narbonne, 31062 Toulouse - France
{ober,bruel}@irit.fr

Abstract. In recent years, numerical simulation has attracted increasing interest within industry and among academics. Paradoxically, the development and maintenance of high performance scientific computing software has become more complex due to the diversification of hardware architectures and their related programming languages and libraries.

In this paper, we share our experience in using model-driven development for numerical simulation software. Our approach called MDE4HPC proposes to tackle development complexity by using a domain specific modeling language to describe abstract views of the software. We present and analyse the results obtained with its implementation when deriving this abstract model to target Arcane, a development framework for 2D and 3D numerical simulation software.

1 Introduction

Thirty-five years ago, Gordon Moore, in one of the most visionary computer-related predictions [1], said that computer performance would increase by 40% per year. That prediction still stands. While for about 30 years that increase in performance was achieved by keeping the traditional sequential programming model, the performance increase has more recently occurred through parallel computer architectures. Such a shift has led to the need to rethink traditional software development in terms of how best to exploit these new architectures.

One of the main concerns of the high-performance scientific computing developer community is to produce efficient code for numerical simulation. Due to their thirst for computational power, this shift had to be initiated a long time ago in order to exploit the architectures of supercomputers. Unfortunately, in current practice mainstream parallel programming models, and in particular those addressing HPC, are low level and machine specific.

Even though good performance levels can be achieved with these approaches, drawbacks in terms of architecture dependency, mix-up of concerns and programming complexity occur:

J. Whittle, T. Clark, and T. Kühne (Eds.): MODELS 2011, LNCS 6981, pp. 213–227, 2011.

- *Applications vs. supercomputers lifetime cycle.* In our application domain, the life cycle of supercomputers is five to seven times shorter than the life cycle of scientific applications[2]. CEA's experience has in fact shown that the simulation models and numerical analysis methods associated with our professional problems have a life expectancy of 20 to 30 years and must therefore be maintained over that period, with all the additional problems that come with software maintenance over such a period of time (e.g. team turnover).

  In parallel, through its TERA program [3], the CEA has decided that its main supercomputer has to be replaced every four years in order to increase its computation power by a factor superior to ten (Tera-1: 2002, Tera-10: 2006, Tera-100: 2010). At a pace faster than Moore´s law [1] hardware technological breakthroughs in hardware inevitably appear and software migration problems become an important issue.
- *The lack of separation of concerns.* The problem to be solved - the scientific knowledge of the physics - is entirely mixed with numerical schemes and target dependent information, added to manage the parallelism. Once a complex system has been built, it is difficult to extract the physical models. As a result, maintenance and upgrading become even more complicated.
- *Inaccessibility to domain experts.* The complexity of software programming restricts the use of these workstations and supercomputers to a few scientists who are willing to spend a significant amount of time learning the specificities of a particular set of machines.

Furthermore, the situation is getting worse with the new emerging generation of machines: hybrid machines. They are built by mixing heterogeneous hardware resources such as CPUs with many cores, Graphics Processing Units or CELLs[4]. GPUs are usually found within graphics cards, where they compute the rendering of massive 2D and 3D scenes. However, hardware manufacturers of supercomputers have started to integrate GPUs, since they are particularly well suited to specific operations such as matrix computations and thus linear algebra solving. GPUs contain a large number (in the range of hundreds) of stream processors which increase the computation power of supercomputers. To exploit them, however, developers have to depend on hardware manufacturer specific instructions (NVIDIA Cuda [5], or in the best case, on libraries which attempt to be more generic such as the OpenCL API[6]).

We think that model-based development techniques such as MDA [7] can help us deal with this complexity. In accordance with this opinion, we described in [8] the characteristics and possibilities of such a development approach. In this paper we present results of experiments conducted using this approach.

The rest of this paper is organised as follows: in Section 2 we complete the presentation of the MDE4HPC approach introduced in our previous paper. In Section 3 we introduce ArchiMDE, an implementation of the MDE4HPC approach as well as results obtained using this tool for the development of a numerical simulation software. Finally in Section 4 we discuss the contributions of our research and give directions for future work.

2 MDE4HPC

The Model-Driven Engineering for High Performance Computing (MDE4HPC) approach aims to offer solutions for the development of scientific computing software. The foundations of this approach were presented in [8]. This section aims to complete this broad description by detailing concepts required for the understanding of the results presented in Section 3.

2.1 Collaborative Approach

The development of a numerical simulation software requires the completion of a variety of tasks. Several skills are involved in this process, of course depending on the size of the project and hence the team, while certain tasks might be assigned to only one person.

We think that model sharing between persons from different areas of expertise is a key feature in faster development as it enables enabling reuse, traceability and consistency of the information. The different expertise profiles involved in the development of numerical simulation software and their viewpoint on the global model are presented in Figure 1. This Figure shows that the user point of view on the model of the simulation software is different according to the task he has to perform.

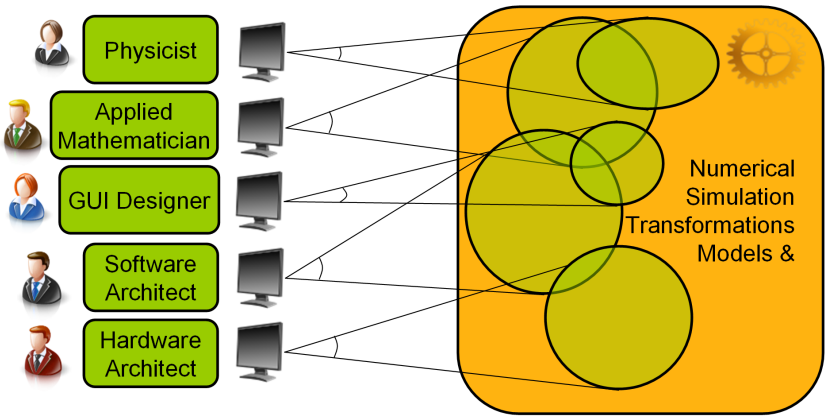

Fig. 1. Viewpoints in scientific computing

2.2 HPCML

High Performance Computing Modeling Language (HPCML) is a domain specific modeling language designed for the description of numerical simulation software. Its specification is part of the MDE4HPC approach. Figure 2 presents a simplified view of the concepts available in the HPCML metamodel for PIM (Platform Independent Model) modeling of the static aspects. Some of these concepts are

intentionally derived from technologies massively used in the scientific computing community, such as Fortran. In fact, we wanted to raise the level of abstraction during the development process without revolutionizing development habits.

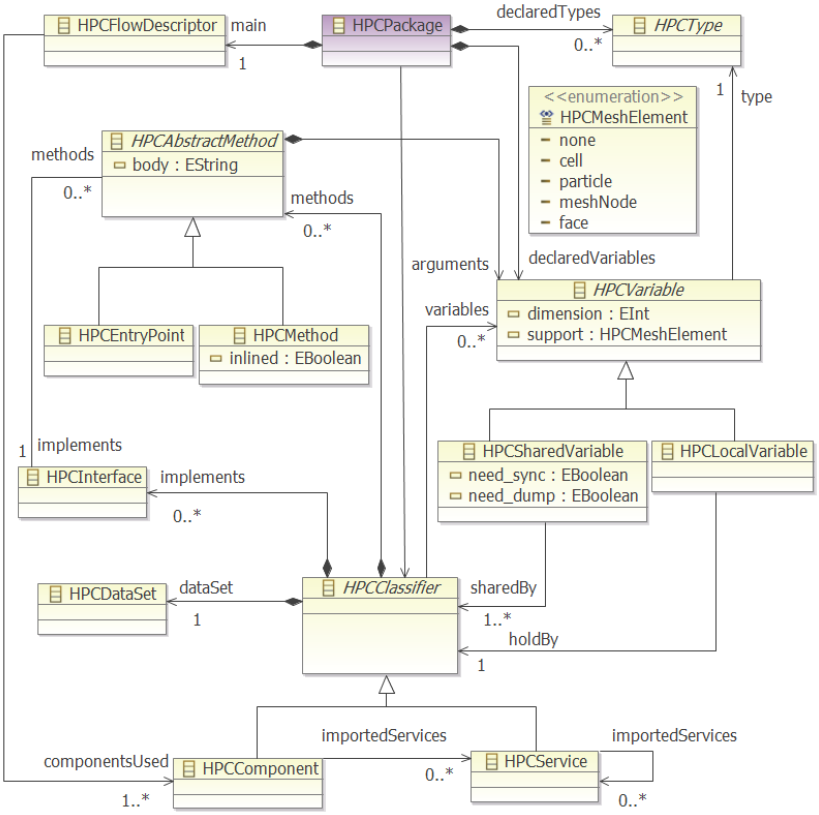

Fig. 2. Simplified view of the HPCML PIM metamodel

The basic building block of HPCML is the *HPCClassifier*. This structural block enables the description of a set of methods which work on a collection of *HPCVariable* that are shared between them. Usually the goal of a numerical simulation is to forecast the evolution in time and space of one or several physical phenomena. In concrete terms, each step of a loop makes the simulation go forward in time which is why this loop is sometimes called a time loop. Computation is performed until the loop stop condition is satisfied (evolution time, physical state reached...). An *HPCFlowDescriptor* describes the sequence of methods which composes the application and thus possesses specific constructs to model this kind of loop.

Within abstract models, we choose to adopt a data parallelism approach based on domain decomposition. Variables can be associated with a mesh element

(vertex,face,cell,particle). This information will guide the concrete implementation of data organization. Shared variables (*HPCSharedVariable*) are also an important modeling element as they allow us to express parallelism between different components.

Even though refinement transformations do not take this information into account in the current version of the tool, it is possible to model high level task parallelism within *HPCFlowDescriptor* via fork/join constructs.

3 Experiment Results

This section presents the results of an experiment conducted with the tool *ArchiMDE*, an implementation of the MDE4HPC approach. Before setting out the results, we first introduce projects in relation with *ArchiMDE* and present their integration within the overall development process.

3.1 Paprika Studio

The specification of a complete and coherent dataset from a numerical simulation has always turned out to be a complex task for the end user. For years, human input has been necessary to fulfil this task, usually provided by the developer of the simulation as the person with the best knowledge of the algorithms parametrization. To reduce the degree of involvement of the developers and to expand the community of end users, graphical user interfaces were introduced by specialists.

These specific editors integrate hard coded rules for managing the inputs of the scientific dataset which are specified by the simulation software developer. This co-development method allows the end user—assuming an exhaustive phase of manual validation—to produce complete and coherent datasets for the application. However, the dispersion of knowledge between the HPC application and its user interface is a real challenge for long term maintainability and traceability, especially when the life time of a simulation software—in the order of several decades—is compared to the frequency of renewal of software technologies for user interfaces. Given that a software simulation and its dataset must be upgraded at the same pace, the maintenance of the editor implies the availability of dedicated skills.

At the CEA, the increasing number and diversity of scientific simulation applications are outpacing the renewal of financial and human resources available for GUI development. Both to meet the goal of strengthening the coherence between a simulation software and its dataset editor and to preserve the separation of concerns, a model-driven approach was adopted for the development of these dataset editors.

Paprika is a software suite based on Eclipse for building scientific dataset editors through the use of model-driven engineering techniques. It includes two essential activities: the modeling of a scientific dataset (Numerical metamodel) and the construction of a graphic editor based on dataset and GUI models (GUI metamodel).

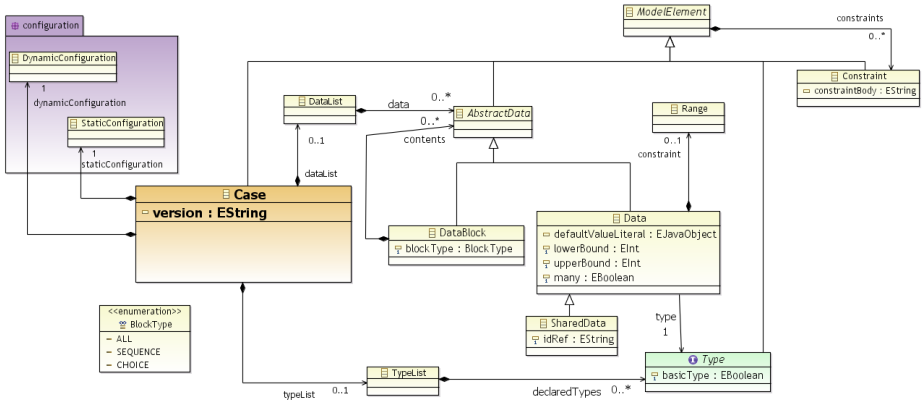

Fig. 3. Simplified view of the Numerical metamodel

In the context of this paper, details about the GUI metamodel are not essential. In consequence we focus our explanation on the Numerical metamodel. A simplified view of its metamodel is presented in Figure 3 and shows that at the highest level, two major concepts are provided:

- the data types, to meet the needs of factorization and reuse of data between several datasets. These types are of three kinds: *predefined*: integer, real, boolean, character string, enumerations; *simple*, i.e. extending a predefined type, for example the "Angle" type by extension of the "real" predefined type; or *structured* to form compound type as from other types. A range value can be specified for a simple data type: default value, minimal and maximal values, increment. It is also possible to associate a simple data type with a physical quantity, for example a frequency, and to set the unit used by default. The structured data types are constructed by the aggregation of predefined and/or simple types. Two structured types can be linked by an inheritance relationship (specialization of a type) or by a reference relationship, with or without containment.
- the data, to define the dataset model. Data are always attached to a predefined, simple or structured type. The supply of predefined types by the *numerical* metamodel makes it possible to define data directly without necessarily defining types beforehand. Data may be isolated or grouped with other data in recursive data blocks.

The choice of Paprika for our experiment was natural for three reasons. Firstly, in our quest for abstraction we needed to model the inputs dataset of the numerical simulation and the Paprika *Numerical* metamodel was already fulfilling that task. Secondly by choosing the *Numerical* metamodel we benefited from the whole generation process to obtain the associated dataset editor. Moreover, as an independent product, Paprika was not capable of generating the persistence management of the dataset and this step was still manual, hence error-prone

and time consuming. But with its integration with ArchiMDE, the automation of this step was feasible, allowing easier software maintenance. Finally, Paprika is developed within our laboratory, so it was easier to access information concerning its architecture.

3.2 The Arcane Framework

In line with the TERA program presented in the Section 1, CEA/DAM's main supercomputer is replaced every four years with a growth of its computation power by a factor superior to ten. In order to prepare for these frequent upgrades, in 2000 the CEA-DAM started the development of Arcane [9], a development framework for 2D and 3D numerical simulation software. Several requirements determined the design of Arcane:

- the management of as many technical details (mesh management, memory management, input/output, parallelism) as possible by the framework itself to simplify software development.
- the possibility to obtain high level of performance on clusters of more than 10000 cores.
- to speed up the development phase by providing a set of tools for building, debugging, verifying and validating numerical software.

In addition to mathematical algorithms for solving physics equations, a numerical simulation has to handle several technical aspects such as the mesh management mentioned previously. However for this experiment we wanted to focus on the definition of the high level concepts without having to deal with a too complex generation chain. That is why we chose to rely on the Arcane framework to manage all those technical aspects as it has shown great capabilities on supercomputers and workstations over the last decade.

3.3 Development Process

The MDE4HPC approach presented in Section 2 proposes to offer a tailored perspective of the project for each kind of participant in the development. ArchiMDE follows this recommendation by providing a set of views, each adapted to a specific task. Figure 4 illustrates the different models and transformations which are part of the development process. In this process physicists and applied mathematicians are responsible for modeling what the numerical core of the software must compute(*HPCML PIM*) and its inputs (*Numerical*). Software engineers and hardware architects are in charge of defining *HPCML PDM* and the rules to combine PIM and PDM models, as well as the rules to refine the PSM model until text based generation (numerical software, GUI, test, documentation...). The work of software engineers and hardware architects on this phase of the process is widely reusable between projects while the target machine does not change. The GUI designer has to model the user interface by deriving the *Numerical* model.

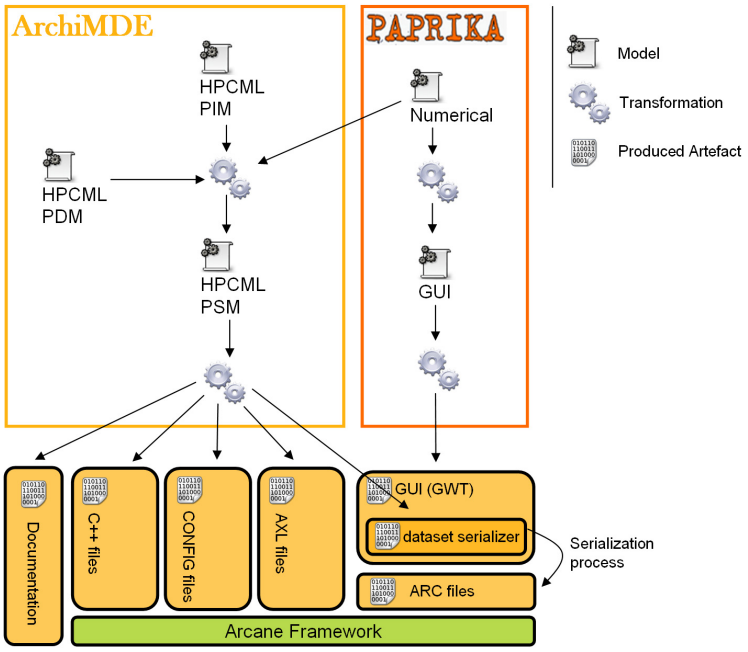

Fig. 4. Development process

All the model transformations are based on Eclipse projects from the *OpenArchitecture Ware* framework. Model-to-Model transformations use the Xtend project and Model-to-Text transformations use the Xpand project. Xpand with its polymorphic template invocation fulfilled most of our needs and its aspect oriented programming possibilities offer a maintenance improvement of M2T transformations. Even though the Xtend syntax and use for M2M transformations is disconcerting at first compared to other M2M framework, it was sufficient for our experiment. Nevertheless the possibility to define functional extensions accessible both from Xpand and Xtend was a powerful and useful feature.

Paprika was not initially designed to be integrated with other modelers such as ArchiMDE. Hence in order to accomplish in ArchiMDE the transformation which takes *Numerical* models as input, we had to define a static mapping between primitive types from Paprika and ArchiMDE. Apart from this point the transformation which integrates the *Numerical* model from Paprika into ArchiMDE is straightforward. *HPCComponents* and their corresponding datasets model are matched together regarding their name.

3.4 Results with an Lagrangian Hydrodynamic Simulation

To assess the validity of the approach, we developed with ArchiMDE a simplified Lagrangian hydrodynamic module introduced in [9] where the mesh nodes are

moved according to Newton's law and the thermodynamic values are updated. At each time step this numerical simulation performs the following operations:

- compute pressure force on nodes:

$$\vec{F}_s^n = \sum_q p_q^n \cdot \vec{C}_q^s$$

- apply dynamic principle and compute node speed:

$$\vec{u}_s^{n+\frac{1}{2}} = \vec{u}_s^{n-\frac{1}{2}} + \frac{\Delta t}{M_s}\vec{F}_s^n$$

- apply boundary conditions.
- move nodes:

$$\vec{x}_s^{n+1} = \vec{x}_s^n + \Delta t \cdot \vec{u}_s^{n+\frac{1}{2}}$$

- update geometric values (meshes volume, meshes characteristic length and geometric components required for the pressure gradient calculation)
- update density:

$$\rho_q^{n+1} = \frac{m_q^{n+1}}{\nu_q^{n+1}}$$

- apply equation of state to update internal energy, pressure and sound speed:

$$e^{n+1} = \frac{1 + \frac{(\gamma-1)}{2} \cdot (1 - \frac{\nu^{n+1}}{\nu^n})}{1 + \frac{(\gamma-1)}{2} \cdot (1 - \frac{\nu^n}{\nu^{n+1}})}$$

$$p^{n+1} = (\gamma - 1)\rho^{n+1}e^{n+1}$$

$$c^{n+1} = \sqrt{\frac{\gamma p^{n+1}}{\rho^{n+1}}}$$

- compute the new time step according to the CFL (Courant-Friedrichs-Levy) constraint.

The *HPCFlowDescriptor* describing the sequence of methods is shown in Figure 5. Listing 1.1 shows the body of the *HPCEntryPoint computePressureForce*. It is an Arcane source code, i.e. C++ syntax plus primitives from the Framework. It is interesting to note Arcane primitives for mesh manipulation (ENUMERATE_CELL) which are at a higher level of abstraction than the usual array manipulation.

The graphic user interface of the produced dataset editor is shown in Figure 6. The version presented here is based on GWT (Google Web Toolkit) but Paprika offers also the possibility to generate from the GUI model a version based on SWT.

Regarding the size of the experiment, the generated source code (computational core and dataset editor) is around 12 KLOC, given that the Arcane source code is relatively compact as many aspects of the numerical simulation are handled by the framework (pre-processing, inputs/outputs management, load balancing, post-processing).

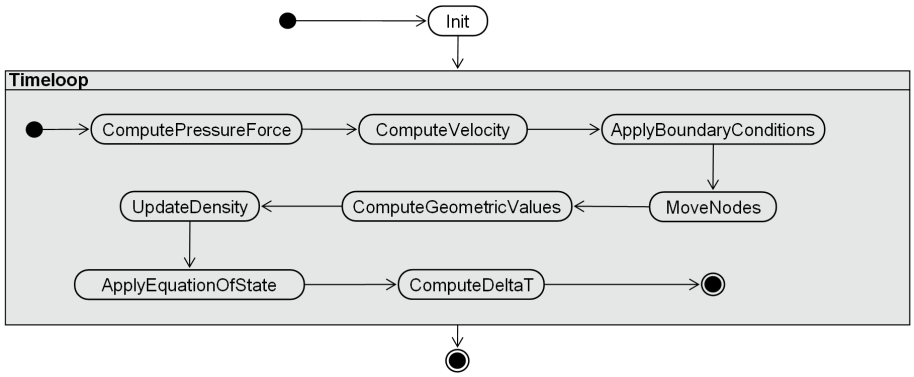

Fig. 5. *HPCFlowDescriptor* of the hydrodynamic simulation

```
// Reset of the force vector
m_force.fill(Real3::null());

// Computation for each vertex of each cell of the
// contribution from the pressure forces
ENUMERATE_CELL(icell, allCells())
{
    const Cell & cell = *icell;
    Real pressure = m_pressure[icell];
    for (NodeEnumerator inode(cell.nodes()); inode.hasNext();
        ++inode)
    {
        m_force[inode] += pressure * m_cell_cqs[icell][inode.
            index()];
    }
}
```

Listing 1.1. Body of the *HPCEntryPoint computePressureForce*

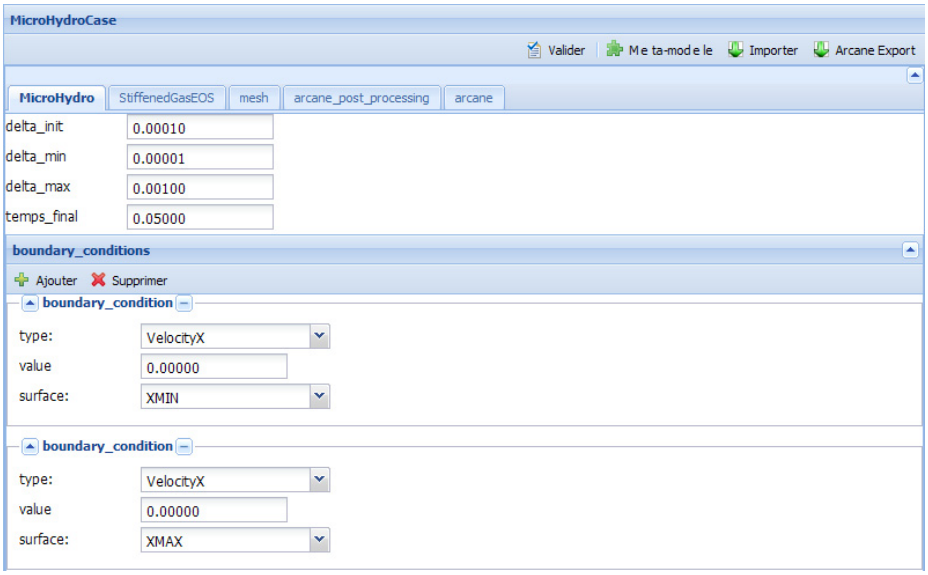

Fig. 6. GUI of the dataset editor generated

We now examine the results of our experiment according to the following points:

- *Performance*. The generated source code of the numerical part (Arcane source code) is similar to the one presented in [9]. Benchmarks of the Arcane framework on the Tera-10 supercomputer are available on this article. Thus from a computational performance point of view both versions would obtain the same results, and benchmarks would only have evaluated the performance of the Arcane framework and not of our approach. However, these results could be improved with the integration of optimization good practice as model transformations.
- *Development Time*. The development time with both approaches —modeling and hand written code— was practically identical. The modeling approach was a little faster thanks to the GUI part. The time taken to develop the different refinement transformations were considered apart from the time required to develop applications. Indeed, as specified in the approach, the transformations rules would be used by several developments, hence their cost could be negligible compared to the application development time. In terms of development productivity, the gain expected by the approach does not appear clearly in this experiment, because the Arcane framework is already at a reasonable level of abstraction and enables developers to avoid certain time-consuming and repetitive tasks. With a low level target generation such as MPI (Message Passing Interface) [10] or Cuda[5], better results would have been obtained.

– *Maintenance.* The approach would reveal its potential with several applications and especially over time when application migrations (adaptive maintenance) will have to be performed. In that case, the benefits would be clearer, as application models could be reused. The gain would therefore be proportional to the number of applications to migrate.

  In the case of upgrade maintenance we will take a specific use case to support our explanation. The scenario is the following: to increase the simulation precision of one software, a new parameter must be added. To accomplish this change, several actions have to be performed: the algorithm of the numerical simulation must be updated to profit from this new parameter and this evolution has to be validated, database validation tests datasets must be migrated, the dataset reader of the simulation must be updated to read this new value, tests must be written to ensure that the reading process is correct, the development documentation and user guide must be updated to explain the role of the new parameter, the graphic user interface of the dataset editor must be updated to display the new value and finally the persistence management of this value must be added into the dataset editor. Table 1 gives for each of these activities, the average time in hours required to perform them with four different approaches. It is always a complex task to measure productivity gains in software development, especially when the sample size available for the experiment is small. These measures are based on the experience of two expert engineers and two trainee engineers. Hence we are working in relative and not absolute terms: the aim is only to observe trends.

Table 1. Average upgrade maintenance time in hour with four different approaches

	Fortran Tcl/Tk	Fortran Paprika	Arcane Paprika	ArchiMDE Arcane Paprika
algorithm update	8	8	5	4.5
algorithm validation	4	4	4	4
validation testing migration	1	1	1	1
dataset reader update	0.7	0.7	0.3	0.15
GUI tests	0.3	0.3	0.1	0
documentation	0.25	0.25	0.2	0.15
dataset editor	4	1	1	1
data persistence	2	1	1	0
Productivity improvement	reference	*1.25*	*1.61*	*1.88*

The improvement from using Paprika instead of Tcl/Tk comes from the fact that GUIs usually contain plenty of simple and redundant source code and that thanks to Paprika this GUI is generated and the repetitive tasks are now replaced by a faster modeling phase. The improvement from using Arcane/Paprika instead of Fortran/Paprika can be explained for two reasons: Arcane offers services to reduce the amount of code to produce and

Arcane provides high level concepts to simplify the development. Finally the improvement with the global solution (ArchiMDE/Arcane/Paprika) comes from the integration (more parts can be generated thanks to information sharing via model transformations) and the higher level of abstraction. Sadly the job which globally benefits the most from the productivity increase is the software engineer.

4 Discussion and Perspectives

This experiment is a further step toward the use of model-based techniques for numerical simulation development in an industrial context. Feedback from this experiment and from other projects such as [11] shows that regarding the GUI, model-based development greatly increases the productivity at a low cost. Regarding the computational part, the Arcane framework is capable of providing us with excellent performances and a good scalability which is our primary objective. In this paper we show that on the one hand performance and productivity are provided by the framework and that on the other hand costs reduction and application durability are provided by MDE. Hence this combination of the two allowed us to reach all of our objectives.

As discussed previously, the use of our approach has allowed us to raise the level of abstraction with respect to existing practices. Basing our approach on an existing framework, Arcane, is a pragmatic choice that allowed us to deploy our approach more rapidly. Nevertheless, the drawback of this choice is that we inherit the limitations of this framework, for instance with respect to a new hybrid architecture. This is why the next step of our research is to extend HPCML to cover the full modeling of the numerical code.

Regarding related work, we can mention the High Productivity Computing Systems (HPCS) programme launched in 2002 by the DARPA [12] from emerged novel programming language: Chapel (Cray), Fortress(SUN) and X10 (IBM). The principal drawback of this approach is the need to develop high-performance compiler, debugger and implementation for each existing architecture. Macro-based approaches such as HMPP (Hybrid Multi-core Parallel Programming environment) [13] and OpenMP (Open Multi Processing) [14] offer a respectable solution for improving legacy code. However, as their use is based on compiler directives which limit the separation of concerns, this solution can appear as less attractive for new developments. The step forward embedded DSL techniques such as [15] is language virtualization as defined in the Lizst project [16]. This project shares the same philosophy as ours and represents a good perspective for the modeling of dynamic aspects in the MDE4HPC approach. Globally we can mention that none of these projects are opposed to our approach as they could be used as target technology at various level of our refinement process.

With the adoption of a full model based development, new possibilities will be offered to us. For example, hybrid machines presented in Section 1 would become accessible in order to increase the performance level. Still to fulfil HPC primary objective, low level optimizations could be achieved via model transformations,

to achieve better performance levels. Furthermore, with conventional low level hand-written source code, the development of multiple versions of the software to assess which one suits the targeted platform best in terms of performance the targeted platform would be too costly. Even though this feature is not yet implemented, we think that higher-order transformations can make this kind of parametric studies accessible. In the same spirit, projects such as StarPU [17] require the algorithm to be implemented in different languages. StarPU is a unified runtime system that offers support for heterogeneous architectures (CPU, GPUs, IBM Cell) by selecting at runtime the more relevant implementation. With our approach, once the generators for each language have been built, the cost of multi-languages generation is extremely low comparing to the hand-written approach.

The validation phase represents a substantial part of the development time, but for the moment only small productivity gains are offered by our approach on this aspect. We plan to include validation tests in the modeling process in order to automatize the migration of database validation tests datasets.

References

1. Moore, G.E.: Cramming more components onto integrated circuits. Electronics 38(8), 114–117 (1965)
2. Lugato, D.: Model-driven engineering for high-performance computing applications. In: The 19th IASTED International Conference on Modelling and Simulation, Quebec City, Quebec, Canada (May 2008)
3. Gonnord, J., Leca, P., Robin, F.: Au delà de 50 mille milliards d´opérations par seconde! La Recherche (393) (January 2006)
4. Johns, C.R., Brokenshire, D.A.: Introduction to the cell broadband engine architecture. IBM Journal of Research and Development 51(5), 503–520 (2007)
5. Kirk, D.: Nvidia cuda software and gpu parallel computing architecture. In: ISMM, pp. 103–104 (2007)
6. KhronosGroup: The OpenCL specification. Technical report (2009)
7. Miller, J., Mukerji, J.: Mda guide version 1.0.1. omg/2003-06-01. Technical report, OMG (2003)
8. Palyart, M., Lugato, D., Ober, I., Bruel, J.M.: MDE4HPC: An approach for using Model-Driven Engineering in High-Performance Computing. In: 15th System Design Languages Forum, SDL 2011 (2011)
9. Grospellier, G., Lelandais, B.: The Arcane development framework. In: POOSC 2009. ACM, New York (2009)
10. Snir, M., Otto, S.W., Huss-Lederman, S., Walker, D.W., Dongarra, J.: MPI: The complete reference. MIT Press, Cambridge (1996)
11. Schramm, A., Preußner, A., Heinrich, M., Vogel, L.: Rapid UI Development for Enterprise Applications: Combining Manual and Model-Driven Techniques. In: Petriu, D.C., Rouquette, N., Haugen, Ø. (eds.) MODELS 2010. LNCS, vol. 6394, pp. 271–285. Springer, Heidelberg (2010)
12. Weiland, M.: Chapel, Fortress and X10: Novel Languages for HPC. Technical report, The University of Edinburgh (October 2007)
13. Bodin, F.: Keynote: Compilers in the manycore era. In: Seznec, A., Emer, J., O'Boyle, M., Martonosi, M., Ungerer, T. (eds.) HiPEAC 2009. LNCS, vol. 5409, pp. 2–3. Springer, Heidelberg (2009)

14. Dagum, L., Menon, R.: Openmp: An industry-standard api for shared-memory programming. Computing in Science and Engineering 5, 46–55 (1998)
15. Christophe, P.: A domain specific embedded language in c++ for automatic differentiation, projection, integration and variational formulations. Sci. Program (2006)
16. Chafi, H., DeVito, Z., Moors, A., Rompf, T., Sujeeth, A.K., Hanrahan, P., Odersky, M., Olukotun, K.: Language virtualization for heterogeneous parallel computing. In: OOPSLA, pp. 835–847. ACM, New York (2010)
17. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.-A.: starPU: A unified platform for task scheduling on heterogeneous multicore architectures. In: Sips, H., Epema, D., Lin, H.-X. (eds.) Euro-Par 2009. LNCS, vol. 5704, pp. 863–874. Springer, Heidelberg (2009), http://hal.inria.fr/inria-00384363/en/